**1、索引**

索引（Index）是帮助MySQL高效获取数据的数据结构。

在数据之外，数据库系统还维护着满足特定查找算法的数据结构，这些数据结构以某种方式引用（指向）数据，这样就可以在这些数据结构上实现高级查找算法。这种数据结构，就是索引。

**优点：**

1、有效缩短数据的检索时间

2、为用来排序或者是分组的字段添加索引可以加快分组和排序顺序

**缺点：**

1、创建索引和维护索引需要时间成本、空间成本

2、会降低表的增删改的效率，因为每次增删改索引需要进行动态维护，导致时间变长

**2、索引的类型**

① 普通索引（单列索引）

② 复合索引（组合索引）

复合索引遵守“最左前缀”原则，即在查询条件中使用了复合索引的第一个字段，索引才会被使用

③ 唯一索引

和普通索引类似，主要的区别在于，唯一索引限制列的值必须唯一，但允许存在空值（只允许存在一条空值）

④ 主键索引

一个表只能有一个主键，不允许有空值。一般是在建表的时候同时创建主键索引

⑤ 全文索引

全文索引主要用来查找文本中的关键字，而不是直接与索引中的值相比较。

**\*\*3、MySQL底层索引数据结构（B树、B+树）附：B\*树**

**1. B树**

一颗m阶的B树（B-tree） 定义如下：

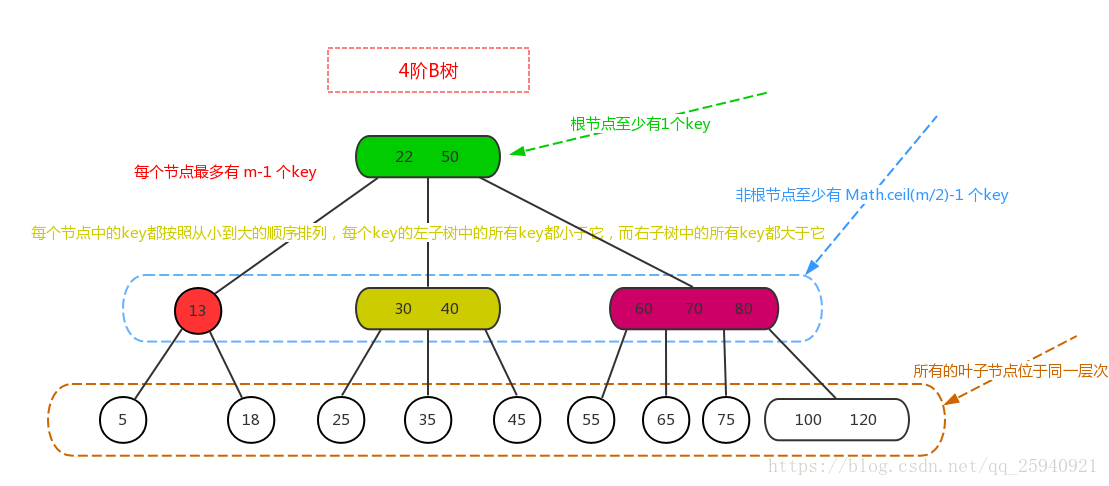
（1）每个节点最多有 m-1 个key；

（2）根节点至少有1个key；

（3）非根节点至少有 (m/2)-1 个key；(向上取整)

（4）每个节点中的key都按照从小到大的顺序排列，每个key的左子树中的所有key都小于它，而右子树中的所有key都大于它；

（5）所有叶子节点都位于同一层，即根节点到每个叶子节点的长度都相同。



B树节点的**插入**

若节点插入节点的key还未“丰满”，则直接插入，若节点插入节点的key已“丰满”，则插入节点之后分裂，再以分裂之后的父节点看作向上层插入的节点调整，直至满足该 m 阶的B树。

B树节点的**删除**

（1）若删除非叶子节点， 找后继节点替换之，将问题转化为删除叶子节点；

（2）若删除叶子节点，且叶子节点的key数大于定义中的最小值，则直接删除即可，无需调整，

（3）若删除叶子节点，且叶子节点的key数刚好满足定义中的最小值，则将节点删除，此时树肯定需要调整，即：

a.若删除节点的相邻兄弟节点的key数大于定义中的最小值，则父节点的1个key下移与待删除的节点合并，相邻兄弟节点的1个key上移与父节点合并，完成调整；

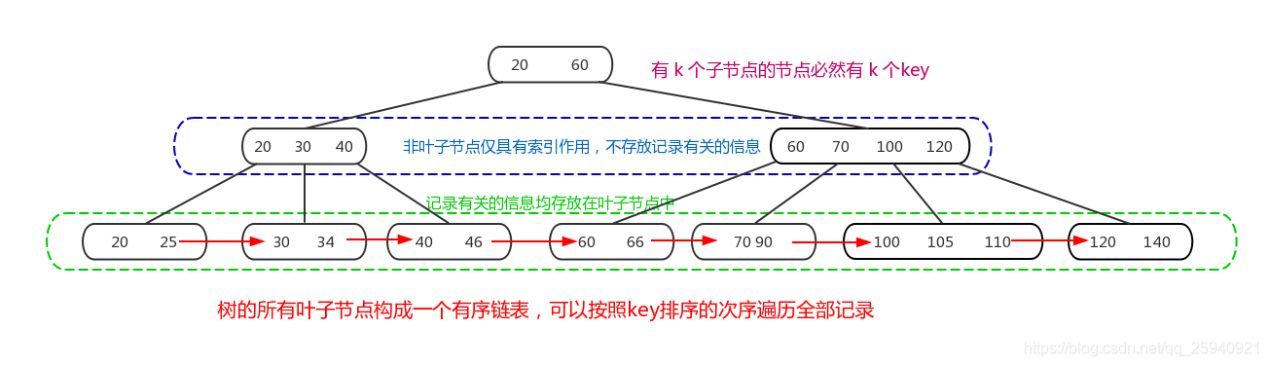
b.若删除节点的相邻兄弟节点的key数刚好满足定义的最小值，则父节点的1个key下移与待删除的节点及相邻兄弟节点，三者进行合并成一个节点，若下移1个key后的父节点的key数陷入“贫穷”，则将父节点看作当前待删除的节点，重复a，b的判断。

**2. B+树**

（1）有 k 个子节点的节点必然有 k 个key；

（2）非叶子节点仅具有索引作用，跟记录有关的信息均存放在叶子节点中。

（3）树的所有叶子节点构成一个有序链表，可以按照key排序的次序遍历全部记录。



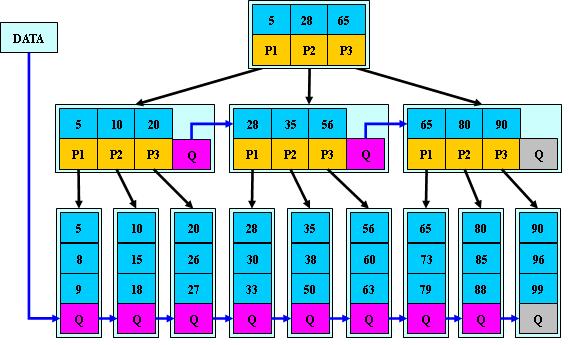
**优点：**

1. 由于B+树在内部节点上不包含数据信息，因此在内存页中能够存放更多的key。数据存放的更加紧密，具有更好的空间局部性。因此访问叶子节点上关联的数据也具有更好的缓存命中率。

2. B+树的叶子结点都是相链的，因此对整棵树的便利只需要一次线性遍历叶子结点即可。而且由于数据顺序排列并且相连，所以便于区间查找和搜索。而B树则需要进行每一层的递归遍历，相邻的元素可能在内存中不相邻，所以缓存命中性没有B+树好。

**3. B\*树**

是B+树的变体，在B+树的非根和非叶子结点再增加指向兄弟的指针；



**4、脏读、不可重复读、幻读**

**-脏读**：某个事务已更新一份数据，另一个事务在此时读取了同一份数据，由于某些原因，前一个RollBack了操作，则后一个事务所读取的数据就会是不正确的。

**-不可重复读**：在一个事务的两次查询之中数据不一致，这可能是两次查询过程中间插入了一个事务更新的原有的数据。

**-幻读**：在一个事务的两次查询中数据笔数不一致，例如有一个事务查询了几列(Row)数据，而另一个事务却在此时插入了新的几列数据，先前的事务在接下来的查询中，就会发现有几列数据是它先前所没有的。

-第1类丢失更新：事务A撤销时，把已经提交的事务B的更新数据覆盖了。

-第2类丢失更新：事务A覆盖事务B已经提交的数据，造成事务B所做的操作丢失。

**\*\*5、MySQL隔离级别**

| **隔离级别** | **脏读** | **不可重复读** | **幻读** |  |
| --- | --- | --- | --- | --- |
| READ UNCOMMITED | 允许 | 允许 | 允许 | 所有事务都可以看到其他未提交事务的执行结果 |
| READ COMMITTED | 不允许 | 允许 | 允许 | 一个事务只能看见已经提交事务所做的改变。 |
| REPEATABLE READ | 不允许 | 不允许 | 允许 | MySQL的默认事务隔离级别，它确保同一事务的多个实例在并发读取数据时，会看到同样的数据行。 |
| SERIALIZABLE | 不允许 | 不允许 | 不允许 | 每个读的数据行上加上共享锁。在这个级别，可能导致大量的超时现象和锁竞争 |

解决脏读：

修改时加**排他锁**(写锁)，直到事务提交后才释放，读取时加**共享锁**(读锁)，其他事务只能读取，不能再有更新操作 。防止脏读。

解决不可重复读：

innodb引擎采用了**mvcc**（多版本并发控制）来解决不可重复读问题。mvcc是利用在每条数据后面加了隐藏的两列（创建版本号和删除版本号）当执行查询的时, 当前查询版本号>= 创建版本号 并且 >删除版本号 , MVCC可以在大多数情况下代替行级锁,使用MVCC,能降低其系统开销。

解决幻读：

Mysql官方给出的幻读解释是：只要在一个事务中，第二次select多出了row就算幻读。

**6、数据库事务**

数据库事务( transaction)是访问并可能操作各种[数据项](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E9%A1%B9/3227309)的一个数据库操作[序列](https://baike.baidu.com/item/%E5%BA%8F%E5%88%97/1302588)，这些操作要么全部执行,要么全部不执行，是一个不可分割的工作单位。事务由事务开始与事务结束之间执行的全部数据库操作组成。

事务的**ACID**属性：

1、原子性：

整个事务中的所有操作，要么全部完成，要么全部不完成，不可能停滞在中间某个环节。 事务在执行过程中发生错误，会被回滚（Rollback）到事务开始前的状态，就像这个事务从来没有执行过一样。

2、一致性：

在事务开始之前和事务结束以后，数据库数据的一致性约束没有被破坏。例如：现有完整性约束 A+B=100，如果一个事务改变了 A，那么必须得改变 B，使得事务结束后依然满足 A+B=100，否则事务失败。

3、隔离性：

事务的执行不受其他事务的干扰，

4、持久性：

对于任意已提交事务，系统必须保证该事务对数据库的改变不被丢失，即使数据库出现故障。

**\*\*7、MySQL存储引擎InnoDB和MyISAM的区别**

1、InnoDB支持事务，MyISAM不支持，对于InnoDB每一条SQL语言都默认封装成事务，自动提交，这样会影响速度，所以最好把多条SQL语言放在begin和commit之间，组成一个事务；

2、InnoDB支持外键，而MyISAM不支持。

3、InnoDB是聚簇索引（数据库表行中数据的物理顺序与键值的逻辑（索引）顺序相同），使用B+Tree作为索引结构，数据文件是和（主键）索引绑在一起的（表数据文件本身就是按B+Tree组织的一个索引结构），必须要有**主键**，通过主键索引效率很高。但是辅助索引需要两次查询，先查询到主键，然后再通过主键查询到数据。MyISAM是非聚簇索引，也是使用B+Tree作为索引结构，索引和数据文件是分离的，索引保存的是数据文件的指针。主键索引和辅助索引是独立的。也就是说：InnoDB的B+树主键索引的叶子节点就是数据文件，辅助索引的叶子节点是主键的值；而MyISAM的B+树主键索引和辅助索引的叶子节点都是数据文件的地址指针。

4、InnoDB不保存表的具体行数，执行select count(\*) from table时需要全表扫描。而MyISAM用一个变量保存了整个表的行数。

5、InnoDB支持表、行(默认)级锁，而MyISAM支持表级锁

①如果是读多写少的项目,可以考虑使用MyISAM,MYISAM索引和数据是分开的，而且其索引是压缩的，可以更好地利用内存。所以它的查询性能明显优于INNODB。压缩后的索引也能节约一些磁盘空间。MYISAM拥有全文索引的功能，这可以极大地优化LIKE查询的效率。

②如果你的应用程序一定要使用事务，毫无疑问你要选择INNODB引擎

**8、聚簇索引和非聚簇索引**

Innodb使用的是聚簇索引，MyISam使用的是非聚簇索引

**聚簇索引**

聚簇索引就是按照每张表的**主键**构造一颗B+树，同时叶子节点中存放的就是整张表的行记录数据，也将聚集索引的叶子节点称为数据页。这个特性决定了索引组织表中数据也是索引的一部分，每张表只能拥有一个聚簇索引。

Innodb通过主键聚集数据，如果没有定义主键，innodb会选择非空的唯一索引代替。如果没有这样的索引，innodb会隐式的定义一个主键来作为聚簇索引。

**非聚簇索引**

将数据存储于索引分开结构，索引结构的叶子节点指向了数据的对应行，myisam通过key\_buffer把索引先缓存到内存中，当需要访问数据时（通过索引访问数据），在内存中直接搜索索引，然后通过索引找到磁盘相应数据

**\*\*9、MVCC多版本并发控制**

多版本并发控制（MVCC）是一种用来解决不可重复读的并发控制，MVCC为事务分配单向增长的时间戳，为每个修改保存一个版本，版本与事务时间戳关联，读操作只读该事务开始前的数据库的快照。

一个支持MVCC的数据库，在更新某些数据时，并非使用新数据覆盖旧数据，而是标记旧数据是过时的，同时在其他地方新增一个数据版本。因此，同一份数据有多个版本存储，但只有一个是最新的。

InnoDB的MVCC，通过在每行记录后面保存两个隐藏的列来实现：一个保存了行的创建时间，一个保存行的过期时间（删除时间），当然，这里的时间并不是时间戳，而是系统版本号，每开始一个新的事务，系统版本号就会递增

**select操作**

InnoDB只查找版本早于（包含等于）当前事务版本的数据行。可以确保事务读取的行，要么是事务开始前就已存在，或者事务自身插入或修改的记录。

行的删除版本要么未定义，要么大于当前事务版本号。可以确保事务读取的行，在事务开始之前未删除。

**insert操作**

将新插入的行保存当前版本号为行版本号。

**delete操作**

将删除的行保存当前版本号为删除标识。

**update操作**

变为insert和delete操作的组合，insert的行保存当前版本号为行版本号，delete则保存当前版本号到原来的行作为删除标识。

由于旧数据并不真正的删除，所以必须对这些数据进行清理，innodb会开启一个后台线程执行清理工作，具体的规则是将删除版本号小于当前系统版本的行删除，这个过程叫做purge。

**10、binlog,redolog,undolog都是什么，起什么作用**

**redo log日志模块**

redo log是InnoDB存储引擎层的日志，又称重做日志文件，用于记录事务操作的变化，记录的是数据修改之后的值，不管事务是否提交都会记录下来。在实例和介质失败（media failure）时，redo log文件就能派上用场，如数据库掉电，InnoDB存储引擎会使用redo log恢复到掉电前的时刻，以此来保证数据的完整性。

**binlog日志模块**

binlog是属于MySQL Server层面的，又称为归档日志，属于逻辑日志，是以二进制的形式记录的是这个语句的原始逻辑，

1、redo log是属于innoDB层面，binlog属于MySQL Server层面的，这样在数据库用别的存储引擎时可以达到一致性的要求。

2、redo log是物理日志，记录该数据页更新的内容；binlog是逻辑日志，记录的是这个更新语句的原始逻辑

3、redo log是循环写，日志空间大小固定；binlog是追加写，是指一份写到一定大小的时候会更换下一个文件，不会覆盖。

4、binlog可以作为恢复数据使用，主从复制搭建，redo log作为异常宕机或者介质故障后的数据恢复使用。

**回滚日志（undo log）**

保存了事务发生之前的数据的一个版本，可以用于回滚，同时可以提供多版本并发控制下的读（MVCC），也即非锁定读

nnodb事务日志包括redo log和undo log。redo log是重做日志，提供前滚操作，undo log是回滚日志，提供回滚操作。

undo log不是redo log的逆向过程，其实它们都算是用来恢复的日志：

1、redo log通常是物理日志，记录的是数据页的物理修改，而不是某一行或某几行修改成怎样怎样，它用来恢复提交后的物理数据页(恢复数据页，且只能恢复到最后一次提交的位置)。  
2、undo用来回滚行记录到某个版本。undo log一般是逻辑日志，根据每行记录进行记录。

**11、sql优化**

**优化之一 - 从数据库设计方面考虑**

表与表之间的业务联系要明确

表字段尽量使用数值型：

属性尽量使用定长

建立合理的索引

**优化之二 - 从SQL语句优化方面考虑**

尽量将要输出的字段写出来（不要SELECT \*）

合理使用连表查询

查询的时候要注意是否走索引

使用索引遵循最佳左前缀特性

**优化之三 - 读写分离与分库分表**

**11、MySQL有哪些锁**

**按锁粒度分类**

**1、行级锁**

行级锁是mysql中锁定粒度最细的一种锁。表示只针对当前操作的行进行加锁。行级锁能大大减少数据库操作的冲突，其加锁粒度最小，但加锁的开销也最大。行级锁分为共享锁和排他锁

特点：

开销大，加锁慢，会出现死锁。发生锁冲突的概率最低，并发度也最高。

**2、表级锁**

表级锁是mysql中锁定粒度最大的一种锁，表示对当前操作的整张表加锁，它实现简单，资源消耗较少，被大部分mysql引擎支持。最常使用的MyISAM与InnoDB都支持表级锁定。表级锁定分为表共享读锁（共享锁）与表独占写锁（排他锁）

特点：

开销小，加锁快，不会出现死锁。发生锁冲突的概率最高，并发度也最低。

**3、页级锁**

页级锁是 MySQL 中锁定粒度介于行级锁和表级锁中间的一种锁。表级锁速度快，但冲突多，行级冲突少，但速度慢。因此，采取了折衷的页级锁，一次锁定相邻的一组记录。BDB 支持页级锁。

特点：

开销和加锁时间界于表锁和行锁之间；会出现死锁；锁定粒度界于表锁和行锁之间，并发度一般。

**按锁级别分类**

**1、共享锁**

共享锁又称**读锁**，是读取操作创建的锁。其他用户可以并发读取数据，但任何事务都不能对数据进行修改（获取数据上的排他锁），直到已释放所有共享锁。

如果事务T对数据A加上共享锁后，则其他事务只能对A再加共享锁，不能加排他锁。获准共享锁的事务只能读数据，不能修改数据。

用法：

SELECT … LOCK IN SHARE MODE;

**2、排他锁**

排他锁又称**写锁**、独占锁，如果事务T对数据A加上排他锁后，则其他事务不能再对A加任何类型的封锁。获准排他锁的事务既能读数据，又能修改数据。

用法

SELECT … FOR UPDATE;

**12、SQL语句的执行过程**

一条查询的sql语句先执行的是 FROM负责把数据库的表文件加载到内存中去；WHERE会把所示表中的数据进行过滤，取出符合条件的记录行，生成一张临时表；GROUP BY会把临时表切分成若干临时表；ORDER BY对以上的临时表进行排序；LIMIT取排序后的数据。

**13、数据库三范式**

1NF：每个属性都不可再分

2NF：属性完全依赖于主键。

3NF：属性不依赖于其它非主属性**。**